Towards a product for teaching formal algebra.

Jean-François Nicaud, Denis Bouhineau, Christian Varlet
IRIN, 2 rue de la Houssinière, BP 92208, 44322 Nantes cedex 3, France

Anh Nguyen-Xuan
ESA-CNRS "Cognition et Activités Finalisées", Université de Paris 8,

Abstract. This paper describes a research work whose objective is to help students to learn algebra with a computer. Firstly, we summarise the theoretical foundations and we describe a prototype devoted to the factoring problem. With this prototype, the student solves problems without making the calculations. The student can also observe the system solving a problem and providing explanations. Secondly, we summarise several experiments we carried out with students and we describe the results we got. Last, we present our new objective that consists of making a product that will be available for all students and teachers.

1. Introduction

From 1987 to 1997, the APLUSIX team worked in the field of *Artificial Intelligence in Education*, in the domain of formal algebra, with four research objectives : (1) to develop ILEs (Interactive Learning Environments) that can help students to learn formal algebra, (2) to build theoretical foundations for designing such systems, (3) to do experiments on the systems with students, (4) to analyse how students learn using these systems. We focused on the factorisation problem for developing a full prototype that we experimented with students.

Considering that our system was useful for students and that very few ILEs in algebra were available as products (actually the only one we know is MATPERT [2]), we decided, at the beginning of 1998, to add a new objective to the preceding ones: to make a product, i.e., a system available for all the students and teachers, and we began to work in that direction. Products are generally the objective of companies, not of university researchers. In our domain we think that it is very difficult for a company to create a complex system and to make enough money with it. That is why we decided to make a product ourselves.

In this paper, we summarise the theoretical foundations for designing ILEs in algebra, we describe the last prototype we experimented with students, we summarise the main experiments we conducted and the results we got. Last, we indicate the current state of development of the product and the future work.

2. Modelling human reasoning in formal algebra for building ILEs

We consider domains of knowledge in which problems are solved by transforming algebraic expressions. Examples of such problems are factorisation, equation solving, calculus of primitives. Problems can come directly from mathematics or from various domains (geometry, physic, biology, etc.) where algebra appears as a framework for representing objects and as a tool for solving problems.

This domain of knowledge and problems can be called *cognitive formal algebra*: the formal feature comes from the restriction to *problems that are solved by transforming algebraic expressions*, the cognitive feature comes from the restriction to humanlike reasoning.

2.1. Analysis of human reasoning

We consider features of *cognitive formal algebra* that can be observed at transformation level from the behaviour of students and teachers.

1) The basic mechanism consists of replacing a sub-expression with an expression, using what we call a *transformation rule*. For example, x^2-1 can be replaced by (x+1)(x-1) in 3(x^2-1)+(x+1)(2x+1), providing 3(x+1)(x-1)+(x+1)(2x+1).
2) Many transformation rules come from identities (e.g., A^2-B^2 = (A+B)(A-B)).
3) Transformation rules are matched to sub-expressions.
4) Matching can use expression-concepts (concepts from expressions). For example, matching A^2-B^2 to $4x^2$-1 is not simple because neither does $4x^2$ have the form A^2, nor 1 the form B^2. Human matching uses a concept of square: *as 4 is the square of 2, $4x^2$ is the square of 2x, so that it matches the square of A*.
5) Matching can be made modulo a numerical factor. For example, AB+AC matches (2x-4)(x+1)+(3x-6)(2x-1). Human matching uses a concept of matching modulo a numerical factor: *x-2 is recognised in 2x-4 (modulo 2) and in 3x-6 (modulo 3)*.
6) When they have been used frequently, transformation rules have a compiled form [1]. For example, the rule AB+AC -> (A+B)(A-B) becomes *search a common factor*.
7) Some transformation rules are based on a process, e.g., the discriminant.
8) Rule-concepts (concepts from transformation rules) are used. For example, the transformations of (x-2)[2(x+1)]+(x-2)[3(2x-1)] into (x-2)[2(x+1)+ 3(2x-1)] and of $4x^2$-1 into (2x+1)(2x-1) are called factorisations.
9) There is an important concept of reduction. Reduction rules have generally the highest priority for any problem type.
10) There are well identified problem types (e.g., factorisation, equation solving), however the characterisation of solved forms is not precise and is sometimes teacher dependant.
11) Reduction and cleaning up take a important place in solved forms.

The observation of students and teachers shows that some problem types have no strategic difficulties. At any given time, there is *only one thing to do*. In fact, there are always many applicable transformations. The *only thing to do* is the result of specialised knowledge combi-ning transformation and strategic features, which is another aspect of compiled knowledge [1]. Sometimes, this lack of strategic difficulties is the result of a strong limitation of the domain, like when, at some level, factorisation problems are limited to expressions that match an identity so that they are solved just by applying the identity and reducing the result. For domains with strategic difficulties, it is sometimes necessary to backtrack to a previous step in order to try another path (see figure 1). This means that the solving process must be considered in a heuristic search framework [11] and that knowledge is necessary for deciding when to backtrack, where to backtrack and what rule to choose.

2.2. Modelling human reasoning

We have built a general model of human reasoning in formal algebra for building ILEs, the MCA model (Model based on Compiled knowledge for Algebra), and a particular model of the factorisation domain for building the APLUSIX system. These models implement the features of the above analysis. In the general model, there are a heuristic search framework, a theory of expressions, expression-concepts, matching-concepts, rule-concepts, plans, sub-problems. For details, see [9]. In the particular model for factoring polynomials:

1) There is no plan and no sub-problem because the domain is not complex enough[1].
2) Three expression-concepts have been defined: *monomial*, *factor*, *square*. For example, the concept of monomial has three slots: coefficient, variable and degree. Each of the following expressions: 4, -4, x, -x, 4x, -4x, x^n, $-x^n$, $4x^n$, $-4x^n$ are monomials, and a unique transformation rule is able to calculate the sum of two monomials of the same degree.
3) The matching-concept *modulo a constant factor* is used, in particular for matching factors.
4) Four main rule-concepts have been defined: *strong-factoring*, *strong-expansion*, *strong-reduction*, *elementary*. For example, factoring out a monomial is *strong-factoring* and factoring out a constant is *elementary*.

We have stated the following strategic principle [5]: *For factoring polynomials, use in priority strong-factoring and strong-reduction.* This principle is important for choosing the rule to apply, for deciding when to backtrack and for providing help to the student.

A few models have been built by authors of ILEs for algebra. PRESS [3] has rule-concepts and uses these concepts for choosing the rule to solve equations. But PRESS has no expression-concept. In MATHPERT [2], the essential object is the operator. Operators are chosen with the help of indicators. There is not expression-concept and no real rule-concept. DISSOLVE [10] has expression-concepts, in particular the factor concept, but no rule-concept. These three systems do not use a heuristic search framework.

3. The APLUSIX prototype

From 1987 to 1997, the APLUSIX system evolved. Some functionalities were modified or added, other were suppressed when they seamed to be useless. The last version is described here. The system was mainly used at an intermediate level, and sometimes at a low level. It was particularised for some features for these levels. In both levels, the use of the discriminant was not known by the students.The system provides the learner with two interaction modes, an *observation* mode, in which the student observes the system solving a problem, and an *action* mode, in which the student solves a problem. Moreover, the system may be used in a free way, the student choosing the parameters and the problems, or in a directed way, parameters and problems being described in automata.

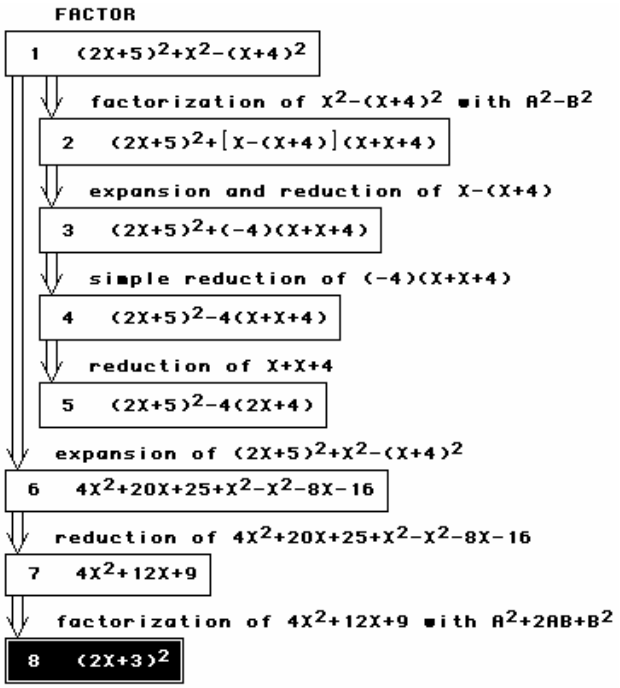

Figure 1: A two branch tree generated by the system.

[1] Plans and sub-problems are implemented for another domain in another prototype.

3.1. The observation mode

In the observation mode, APLUSIX uses *reference knowledge* to solve a problem, step by step, generating and displaying a search tree. For many problems, at the considered level, the search tree consists of a unique branch. However, a two branch tree is sometimes obtained (figure 1). Each step is described using the rule or the rule-concept applied. The rule itself is used for factorisation (the rules under study) otherwise the rule-concept is used. At each step, the learner can ask for explanations about the matching. In this case, the system provides details in a temporary window, for example: *3x-1 is a commun factor in (4-12x)(x-7)-x(9x-3) because (4-12x)(2x-7)=-4(3x-1)(x-7) and -(-2x-7)(9x-3)=-3x(3x-1)*

In previous prototypes, explanations about the strategies were available for the students. They have been abandoned because experiments show that they were useless at this level. They will be available for other levels in the future.

3.2. The action mode

In the action mode, the learner solves problems. At each step, (s)he chooses an action in a menu. The action is either a rule (for factorisations) or a rule-concept (figure 2). After that, an action window is opened for selecting the concerned sub-expression. For factorisations, the student has in addition to indicate the matching (figure 3). The request of the student is then analysed by the system. When it is correct, the system calculates and displays the result. When it is not, a message is displayed. At any time, it is possible to go back to a previous step by clicking on this step. In this mode, the student can ask for help using the menu (figure 2).

Figure 2
The action menu

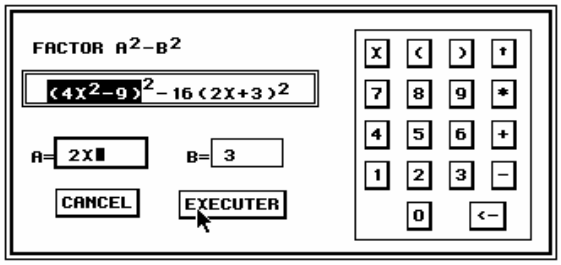

Figure 3: Input of an action

In the *action* mode, the student is freed from calculations and calculation errors. In the action window (figure 3), the input of expressions is controlled and syntax errors are indicated as soon as they occur. The selection of sub-expressions is simplified: the first click selects a monomial, at each other clicks, the minimal sub-expression including all the clicks is selected. Experiments have proved the interest of this mode. It allows the student to focus on the matching and on the strategy so that (s)he can solve more difficult problems. Of course, we do not consider this mode as the unique one to use. It is a complement to the traditional paper/pencil mode.

The analysis function. This function analyses the student's request. Its role is to decide whether the request is correct or not. When the request is correct, it is applied, otherwise feedback is provided. The analysis function does not use the reference solver because there are correct transformations that are not envisaged by the reference solver. For example, *factor out x+2 from (4x+8)(x-1)-(2x+4)(3x-2)* is a correct request that is not generated by the reference solver which generates *factor out 2x+4 from (4x+8)(x-1)-(2x+4)(3x-2)*. The analysis is performed by production rules.

The help function. In the last version, the help function suggests one or two actions among the more promising ones that have not yet been applied. Sometimes, it suggests backtracking. In the first version, the help function was different (we did not want to tell the student what to do): we suggested the possible transformations in the selected step, so that

the student had to think to decide which one to choose. This method did not work. Some students followed the first suggestion every time, others did not use the help because *there were too many things to read*.

3.3. Parameters

Parameters determine the behaviour of the system. In the free mode, the student can change their values. In the directed mode, they are set by an automaton. Some of these parameters are:

- expand-and-reduce: when it is true, the menu includes an expand-and-reduce action allowing expansion and reduction in one step of first order polynomials like 2(x-3)-3(x+1).
- a2+2ab+b2-U: when it is true the identity $a^2+2ab+b^2$ can be used by the student with a value of b such as -2, otherwise, the $a^2-2ab+b^2$ identity is required with b equal to 2.
- help: when the value is true, the action mode allows the student to ask for help.
- delay-help: indicates, when help is true, the delay before giving the student access to help.
- step-by-step: when the value is true, the student is not allowed to perform complex matching for factoring out a polynomial (e.g., for factoring out x-4 from (x-4)(x+1)-(2x-8)(3x-1), (s)he has first to transform 2x-8 in 2(x-4)).

3.4. Automata

An automaton is a list of linked problems in a file. Each problem is described by a name, an expression, an optional modification of the parameters and optional links. For example :

```
(tbcd1  "9-4x^2+(2x-1)(2x+3)"
      (context (mode action) (expand-and-reduce yes) (help yes) (delay-help 5)
            (commute no) (a2+2ab+b2-U no) (do-one-step no) (step-by-step no))
      ((or (> nb-aide 0) (eq erreur factor-out-error) abandon) 1d1)
       (t 1b))
```

says to factor the expression $9-4x^2+(2x-1)(2x+3)$ in the given context. When this problem is finished, the link indicates the next problem.

4. Experiments with the APLUSIX prototype

From 1987 to 1997, we carried out several experiments. The first showed that a wide majority of the students like to work with APLUSIX (as they told us) and learned with the system (this was attested by a pre-test and a post-test). In this section, we summarise the major experiments we carried out during this period.

4.1. Two experiments done in 1990 and 1991

Two pilot experiments were carried out in 1990 and 1991 the objectives of which were, first, to evaluate the user friendliness of the interface, and, second, to determine the degree of difficulty of the different types of matching modulo a numerical factor, and the way perceptual factors may interfere with the recognition of an expression.

The data gathered from these pilot studies led to an important improvement of the interface, particularly in the way the student has to tell the system how to match an expression to a transformation rule. For instance, in the first prototype, when the student wanted to apply the rule A^2-B^2 -> (A+B)(A-B) to the sub-expression $4x^2-9$, (s)he had just to designate the sub-expression and the rule. In the improved interface, the student must designate the sub-expression, the rule and, in addition, the values of A and B. In this way, the student is more active, and it is possible to evaluate the degree of the student's matching knowledge. For instance, a student may recognise that the expression $4x^2-9$ matches the rule A^2-B^2, and that the values of A and B are, respectively, 2x and 3; but (s)he may not recognise that the expression $4(3x-2)^2-9$ matches the same rule (with the correct values for A and B).

Hand analysis of the protocols showed that the matching of an expression to a rule can be more or less difficult, depending on the necessity of implicitly factoring out numerical factors, to obtain an expression that can be directly matched to the rule. This result led us to define the notion of "visibility": an expression is totally "visible" when it matches directly a rule.

4.2. An experiment realised in 1992

A controlled experiment on learning to solve factorisation problems with the *action* mode, was carried out in 1992 with a new version APLUSIX/V0-M2. The student can ask for advice when (s)he do not know how to proceed. When (s)he makes a mistake, the system gives the student a comment about the mistake, an "error message".

This experiment comprised three phases. The first two phases concerned the acquisition of matching skills based on the rule *factoring out* and *second order identities*. In the third phase, the problems were more complex, they can be solved in different ways, and for some of them, the students must choose between different promising rules. A wrong choice may lead to an impasse. Forty-six secondary school students (14-16 years-old) participated in the experiment. Each student spent four forty-five minutes sessions with the system.

A program was developed to diagnose the student's state of matching knowledge during the learning process [6]. The program distinguishes between 17 categories of matching, the categories being defined by five types of basic *visibility* of matching. To evaluate the student's matching knowledge, the method used consists of sequentially increasing or decreasing a student's matching score for a category of matching knowledge. The score for a category of matching knowledge is increased every time (s)he performs *correctly* and *without help* a transformation involving this category.

The individual protocols were given to the program to calculate the evolution of the students' score during the learning experiment. The results showed that the notion of *visibility* we defined was relevant: the less an expression is visible in regard to a given transformation rule, the more it is difficult to be mastered.

A hand analysis of the protocols showed three important points. Firstly, the majority of students learned to backtrack, something they did not do in a paper-and-pencil situation. The search tree presented in the display may have played the role of a reification [4, 12] that helped foster the concept of a search space.

Secondly, observing the way in which matching knowledge was acquired led us to differentiate between two different styles of learning which we have termed *the conceptual based learning style* and *the experiential based learning style*. We hypothesise that these styles of learning are related to the depth of the student's conceptual knowledge of algebra. The student who learned in an experimental way progressed slowly and did not generalise his (her) matching ability immediately from one expression to another : (s)he may have acquired his (her) matching skill solely by establishing a relation of surface similarity between the structures of symbols representing an abstract rule and the structure of symbols representing an algebraic expression. Conversely, students whose progress was based on a deeper understanding of the concepts of *square* and of *factor* had less difficulty in quickly generalising their matching ability over expressions that had different surface features.

Thirdly, we have observed that the matching knowledge about the identities was more stable than the matching knowledge about the *factor out* rule. We hypothesised that this result was related to the way the error messages and the prompts were formulated by the system. In the case of identities, the prompts were general, the system told the student to apply an identity to an expression, but it did not tell the student what were the values of A and B to type in. So that the student still had something to do by him(her)self: find the correct values of A and B. The error messages were, on the contrary, very precise. Indeed, in the case where the student wanted to match an expression E to an identity, and the

expression cannot match, the error message was "this expression is not a difference of two squares". When the student gave an erroneous value of A and/or B for an identity, the system's error message indicated clearly in which way it was an error, by saying, for instance, "With the values of A and B you gave, $A^2-2AB+B^2= x^2-8x+16$ and not x^2-4x+4." So that the student can understand why his (her) action was erroneous.

By contrast to the case of identities, in the case of factoring out a (numerical, monomial, or binomial) factor from an expression, the error messages were laconic, and the prompts too precise. For any error in applying the *factor out* rule, the error message was always the same : "it is not possible to factor out [expression designated as A] from [selected expression]." So that there was no hint about the reason of the error. The prompt concerning the application of the *factor out* rule was, on the contrary, too precise. Consider the student who asks for help at the problem state (4-12x)(2x-7)-(-2x-7)(9x-3). Among the possible actions is the application of the *factor out* rule. The prompt was: "factor out 3x-1 from (4-12x)(2x-7)-(-2x-7)(9x-3)." So that the student did not have anything to do except to copy exactly the prompt.

4.3. An experiment carried out in 1994

An experiment was conducted in May 1994 [7], with 12 students of the same age. The objective of this experiment was to test the hypothesis that less detailed prompts about the application of the *factor out* rule, and more detailed comments about the errors in applying this rule would result in an improved learning of this rule, and would also provide more useful data for the analysis of the learning process. The learning situation was identical to that of the preceding experiment. The prompts relating to the identities were not modified. The prompts for the *factor out* rule were more general. Consider the student who asks for help at the problem state (3x+6)(3-2x)-(2x+3)(-8-4x). The prompt was formulated more laconically "factor out a common factor from (3x+6)(3-2x)-(2x+3)(-8-4x)." So that the student still had to find out by him(her)self the common factor of the expression. The comments on errors differentiate two cases, and the formulation of the comments was more precise about the origin of the error. (i) If the student gives a value of A (a numeral, a monomial or a binomial) that is not a common factor, the system picks out the terms that do not have A as a factor. (ii) If the student wants to factor out a numeral, a monomial or a binomial from a product (for example "factor out 3 from (3x+6)(3-3x)") the system says: "I cannot do what you asked, because *common factor* applies only to sums." The results showed that the improvement in the matching knowledge about the *factor out* rule was statistically significantly better for the subjects in this learning experiment than in the previous one.

4.4. An experiment carried out in 1997

The previous experiments used only the *action* mode. A learning experiment was then conducted in 1997 in which we compared four groups of 20 students. The 80 students were selected by a pre-test in order to obtain four groups that did not differ in their factorisation problem solving ability: low level and high level students were eliminated from the experiment. The 80 students were randomly assigned to one of the four groups in a 2 by 2 factorial plan: two learning contexts (the *learning-with-examples* context and the *learning-just-by-solving* context), and presence or absence of help/explanation from the system. These results suggest that, concerning the matching knowledge, learning by action is more effective than learning by observation [8].

5. Conclusion, current state and future work

5.1. Conclusion

A priori, scientific research does not have the role of realising products, with the industrial and commercial meaning of this term. As an explanation of that situation, we could say that

the development of prototypes is sufficient and satisfactory enough to validate a theory, or to allow experiments. This was the way we worked during the last decade.

From 1987 to 1997, the various prototypes built up for APLUSIX filled these two objectives: (1) the approach adopted and the problem-solving model (the MCA model cited in section 2.2) made it possible indeed to analyse, model and help algebraic manipulation done by students during problem-solving activities in pre-algebra; (2) experiments led to the refinement of the models and trying to answer general questions about learning theories.

But, since the beginning of 1998, the APLUSIX team has decided to design a new version of APLUSIX which will not be a prototype but a product. The work considered for this purpose must tackle constraints of several kinds, which give an explanation of our motivation:

1) constraint of openness: we want to continue the research on the modelling of the cognitive processes of problem-solving and the research on the modelling of the learning processes;

2) constraints of availability: we want to give a general tool (APLUSIX) to teachers and researchers in psychology and didactics to take part in the research (this introduces one of the major characteristics of a product: to be available for all);

3) constraint of quality (design): we want to allow the use and the realisation of experiments on a large scale (feedback is waited for), which introduces other characteristics of a product: robustness, ergonomics;

4) constraints of adaptability and parametrisation: we want to develop the scope of APLUSIX, development in terms of the variety of possible activities (factorisation, equation solving calculus of primitives, etc.) and in terms of available school levels. The parametrisation is intended to allow the product to adapt its functions according to the desired use.

The terms *openness*, *availability*, *quality*, *parameterisation* summarise our objectives. Some of them are clearly the attributes one can hope in a product. The others relate to the research aspect that APLUSIX will continue to have.

5.2. From prototype to product: rewriting rules for the knowledge base

The flexibility offered by a prototype is not conceivable for a product under the same conditions. If one can imagine modifying a prototype to carry out such or such experiment or such adjustment, that is not easily conceivable with a product. Each modification would mean a new version of the product to maintain and commercialise. And yet we want to have the same flexibility with the future product as with the preceding prototypes. In order to obtain such flexibility, we have changed the framework.

One major change is: the product that we are realising is constructed over one knowledge base which is partly external. This part of the knowledge base will be represented by a set of rewriting rules written in a natural form. The knowledge base will be accessible to an advanced user (teacher or experimenter). If one occults the aspects related to the robustness, the availability and maintenance, the choice for a definition of the knowledge base, independent and accessible, is a major difference between the preceding prototypes and the product to come. We count on this major characteristic of the next APLUSIX to ensure the openness and the parametrisation of the product we want to design.

The development of the elements characteristic of the external knowledge base of APLUSIX was based on the preceding experiments and the study of the preceding prototypes. In particular, the study of the preceding prototypes emphasised a set of parameters and a set of knowledge in algebra used, implicitly or not:

- parameters of environment (visibility of rules, availability of help and diagnostic)
- parameters for the application of rewriting rules (search for sub-expressions, conceptual matching, matching modulo a coefficient)
- grouping of rewriting rules in semantic groups (factorisation, reduction, development, etc.)
- establishment of a hierarchy in the application of rewriting rules.

This set of parameters will be defined in the knowledge base. Algebraic knowledge contained in the preceding prototypes was expressed in a compiled form. In the same way, the matching mechanisms specific to algebra were compiled (matching of sub-expressions, conceptual matching, matching modulo a factor). Sometimes these mechanisms were badly distinguished from the rest of the code. From a general point of view, this knowledge was static. They evolved only with the change of the prototype. Furthermore, this knowledge was not accessible in extenso to non-specialists. However a mathematical form, was diffused [9]. It is this form which was used as a starting point for writing the knowledge base of the future product. For example, in [9] three rules like the following expressed factorisation by a common factor:

If E is of the form A1+... +An with n>= 2 **and** each Ai admits the factor U with degree pi
and Q is a canonical integer ranging between 1 and the minimum of pi
and for any i, U with the degree Q is a factor of Ai with cofactor Ci **then** replace E by Uq(C1+... +Cn)

The equivalent formulation that we wish to obtain is the following much more natural one: a*b+a*c -> a*(b+c). The use of this rule is not straightforward. It is done using a complex matching mechanism which takes into account the concepts seen in 2.1.

Note: If the definition of the rules is *opened*, the matching mechanism is *closed*. It is compiled and comprises non trivial mathematical knowledge (associativity, commutativity, identity elements, etc.). However this mechanism is dependent on parameters. These parameters can be changed by program, and defined a priori in the knowledge base for each occurrence of a rewriting rule in one group.

5.3. A product soon

The current state of the work is the following: (1) strategic aspects of the problem-solving are designed, (2) display/selection of algebraic expressions are realised, (3) creation ex nihilo of algebraic and the strategic elements have been initialised, (4) a plain algorithm for matching terms with rewriting rules (with matching of sub-expressions modulo associativity and commutativity) is already obtained, (5) an advanced matching algorithm (with conceptual matching, matching modulo a factor) is nearly achieved. A first version of the final product, with more or less the same functionalities as APLUSIX-M0-V2, is planned for the end of 1999. Other versions, with a developed scope are planned for 2000. Our current platform for the work is Windows 95 and Object Oriented Pascal Delphi.

6. References

[1] Anderson J. R. (1983): The Architecture of Cognition. Harward University Press.

[2] Beeson M. (1996): Design Principles of Mathpert: Software to support education in algebra and calculus, in: Kajler, N. (ed.) *Human Interfaces to Symbolic Computation*, Springer-Verlag.

[3] Bundy A. & Welham B. (1981): Using Meta-level Inference for Selective Application of Multiple Rewriting Rule Sets in Algebraic Manipulation. *Artificial Intelligence, Vol 16 (2)*.

[4] Conlon, T. (1993). PathFinder: A Programming Tool for Search Based Problem Solving. *Proceedings of the Seventh International PEG Conference*, Edinburgh, 74-82.

[5] Gélis J.M. (1994). Elements of a theory of algebra for ILEs. *Proc. of the international symposium on mathmeatics/science education and technology.* San Diego. AACE publisher.

[6] Nguyen-Xuan A., Joly F., Nicaud J.F, Gélis J.M. (1993). Automatic diagnosis of the student's knowledge state in the learning of algebraic problem solving. *Proc. of AI-ED93.*

[7] Nguyen-Xuan, A., Nicaud, J. F. & Gélis, J. M. (1997). Effects of feedback on learning to match algebraic rules to expressions with an intelligent learning environment. *J. of Computer in Math and Science Teaching*, *16*, 291-321.

[8] Nguyen-Xuan, A., Bastide A., Nicaud, J. F (1999). Learning to match algebraic rules by solving problems and by studying examples with an ILE. *Proc. of AI-ED'99.*

[9] Nicaud J.F. (1994): Modélisation en EIAO, les modèles d'APLUSIX. *Didactique et acquisition des connaissances scientifiques*, Vol 14, (1-2), la Pensée Sauvage, Grenoble.
[10] Oliver J., Zukerman I. (1990): DISSOLVE: An Algebra Expert for an ITS. *Proc. of ARCE*, Tokyo.
[11] Pearl J. (1984): *Heuristics*. Addison-Wesley, London.
[12] Singley, M. K. (1990). The reification of Goal Structure in a Calculus Tutor: Effects on Problem-Solving Performance. *Interactive Learning Environments*, *1* (2), 102-123.